TrustedMR: A Trusted MapReduce System based on Tamper Resistance Hardware

Quoc-Cuong To, Benjamin Nguyen, Philippe Pucheral

TrustedMR: A Trusted MapReduce System based on Tamper Resistance Hardware

Quoc-Cuong To, Benjamin Nguyen, Philippe Pucheral

SMIS Project, INRIA Rocquencourt, 78153 Le Chesnay, France
PRiSM Laboratory, 45, Av. des Etats-Unis, 78035 Versailles, France

<Fname.Lname>@inria.fr , <Fname.Lname>@prism.uvsq.fr

Abstract. With scalability, fault tolerance, ease of programming, and flexibility, MapReduce has gained many attractions for large-scale data processing. However, despite its merits, MapReduce does not focus on the problem of data privacy, especially when processing sensitive data, such as personal data, on untrusted infrastructure. In this paper, we investigate a scenario based on the *Trusted Cells* paradigm : a user stores his personal data in a local secure data store and wants to process this data using MapReduce on a third party infrastructure, on which secure devices are also connected. The main contribution of the paper is to present *TrustedMR*, a trusted MapReduce system with high security assurance provided by tamper-resistant hardware, to enforce the security aspect of the MapReduce. Thanks to TrustedMR, encrypted data can then be processed by untrusted computing nodes without any modification to the existing MapReduce framework and code. Our evaluation shows that the performance overhead of TrustedMR is limited to few percents, compared to an original MapReduce framework that handles cleartexts.

Keywords: privacy-preserving, tamper-resistant hardware, MapReduce.

1 Introduction

We are witnessing an exponential creation and accumulation of personal data: data generated and stored by administrations, hospitals, insurance companies; data automatically acquired by web sites, sensors and smart meters; and even digital data owned or created by individuals (e.g., photos, agendas, invoices, quantified-self data). It represents an unprecedented potential for applications (e.g., car insurance billing, carbon tax charging, resource optimization in smart grids, healthcare surveillance). However, as seen with the PRISM affair, it has also become clear that centralizing and processing all one's data in external servers introduces a major threat on privacy. To face this situation, personal cloud systems arise in the market place (e.g., Cozy Cloud[1], SeaFile[2], to cite a few) with the aim to give the control back to individuals on

[1] http://cozy.io/
[2] http://seafile.com/en/home/

their data. According to [29], a Personal Cloud could be defined as a way to aggregate the heterogeneous personal data scattered in different areas into one (virtual) cloud, so that a person could effectively store, acquire, and share his data. This user-centric definition illustrates the gravity shift of information management from organizations to individuals [16]. But this raises a critical question: how to perform big data computations crossing information from multiple individuals?

Trusting a regular Cloud infrastructure to host personal clouds and perform global computations on them is definitely not an option. Privacy violations are legion and arise from negligence, attacks and abusive use and no current server-based approach seems capable of closing the gap[3]. Cryptographic-based solutions have been proposed (e.g., [8, 18, 22]) to guarantee that data never appear in the clear on the servers but they provide either poor performance, poor security or support a very limited set of computations. Consequently, several attempts of personal data management decentralization have appeared (e.g., [1, 3, 17]). While these solutions increase the control of each individual on his data, they complexify big data computations crossing data from several individuals. Solutions have been proposed to solve specific problems like data anonymization [2] or SQL-like queries [21] over decentralized personal data stores. However, data availability can no longer be assumed in this context because individuals can disconnect their personal data stores at their will. Hence the semantics of these computations must be revisited with an open world assumption in mind.

This paper explores a new alternative where individual's data is hosted by a Cloud provider but the individual retains control on it thanks to a personal secure hardware enclave. This alternative capitalizes on two trends. On one side, Cloud providers (e.g., OVH in Europe) now propose to rent private (i.e., unshared) physical nodes to individuals at low cost. On the other side, low cost secure hardware devices like personal smart tokens become more and more popular. Smart tokens have different form factors (e.g., SIM card, USB token, Secure MicroSD) and names but share similar characteristics (low cost, high portability, high tamper-resistance), introducing a real breakthrough in the secure management of personal data [3]. Combining both trends seems rather natural and leads to the infrastructure pictured in Figure 1. This is nothing but a regular Cloud infrastructure with personal secure devices connected to its storage and computing nodes. Hence, each individual could upload his data on the Cloud in an encrypted form and retain the control on it thanks to a Trusted Data Server hosted in his own secure device [1]. Hence the name personal enclave since the Cloud provider has no way to get access to the secrets stored in each secure device nor can tamper with their processing. This architecture differs from [5] where a shared server is hosted in a single tamper-resistant processor. This architecture can be seen as a clustered implementation of the Trusted Cells vision [3], that is to say a set of low power but highly trusted computing nodes which can communicate and exchange data among them through an untrusted Cloud infrastructure to perform a secure global computation.

In this article, we focus on the MapReduce framework [11] to perform big data computations over personal data. With MapReduce, developers can solve various

[3] http://www.datalossdb.org/

cumbersome tasks of distributed programming simply by writing a map and a reduce function. The system automatically distributes the workload over a cluster of commodity machines, monitors the execution, and handles failures. Current trends show that MapReduce is considered as a high-productivity alternative to traditional parallel programming paradigms for a variety of applications, ranging from enterprise computing to peta-scale scientific computing. However, the raw data can be highly sensitive: at the 1Hz granularity provided by the French Linky power meters, most electrical appliances have a distinctive energy signature. It is thus possible to infer from the power meter data inhabitants activities [15]. With the architecture presented in Fig. 1, raw data of each individual could be uploaded in an encrypted form in the Cloud while the cryptographic keys remain confined to the individual's secure device. With appropriate execution and key exchange protocol a global computation can occur with the guarantee that no adversary can get any clear text data nor infer any value at the intermediate steps of the processing.

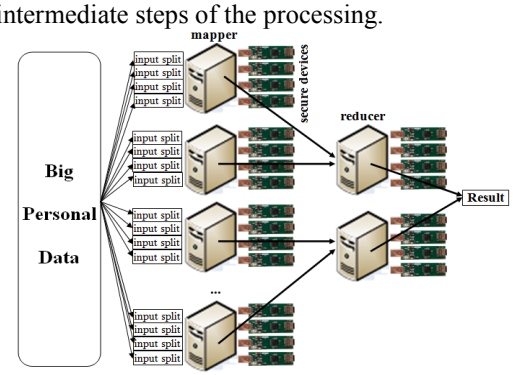


Fig. 1. Cloud infrastructure with personal enclaves

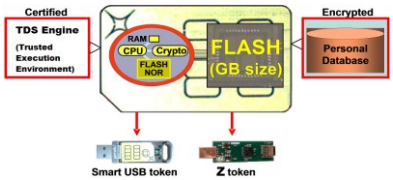


Fig. 2. Secure Device

MapReduce was born to meet the demand of performance in processing big data, but it is still missing the function of protecting user's sensitive data from untrusted mappers/reducers. Although some state-of-the-art works have been proposed to focus on the security aspect of MapReduce, none of them aims at data privacy (see section 2). Based on the architecture presented in Figure 1, this paper proposes a MapReduce-based system, addressing the following four important issues:

1. ***Security***: How to perform a MapReduce computation over personal data without revealing sensitive information to untrusted mappers/reducers nodes?
2. ***Performance***: How to keep acceptable MapReduce performance, that is to say a small overhead compared with processing cleartext data?
3. ***Generality***: How to support any form of Map and Reduce functions?
4. ***Seamless integration***: How to answer the preceding questions without changing the original MapReduce framework?

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 states our problem. Section 4 presents our proposed solution and Section 5 analyses its security. Section 6 measures the performance and section 7 concludes.

2 Related Works

2.1 Security in MapReduce

Mandatory access control (MAC) and differential privacy: [19] proposes the Airavat that integrates MAC with differential privacy in MapReduce framework. Since Airavat adds noise to the output in the reduce function to achieve differential privacy, it requires that reducers must be trusted. Furthermore, the types of computation supported by Airavat are limited (e.g., SUM, COUNT). The other drawback of Airavat is that the security mechanisms are implemented inside the open infrastructure. Hence, their trustworthiness should still be verified. Finally, they have to modify the original MapReduce framework to support MAC.

Integrity verification: In other directions, [23] replicates some map/reduce tasks and assign them to different mappers/reducers to validate the integrity of map/reduce tasks. Any inconsistent intermediate results from those mappers/reducers reveal attacks. However, with only the data integrity, they cannot preserve the data privacy since the mappers/reducers directly access to sensitive data in cleartexts. So, these works are orthogonal to our works in which we aim at protecting the data privacy.

Data anonymization: [26] claims that it is challenging to process large-scale data to satisfy k-anonymity in a tolerable elapsed time. So they anonymize data sets via generalization to satisfy k-anonymity in a highly scalable way by MapReduce.

Hybrid Cloud: Some works [24, 25] propose the hybrid cloud to split the task, keeping the computation on the private data within an organization's private cloud while moving the rest to the public commercial cloud. Sedic [24] requires that reduction operations must be associative and the original MapReduce framework must be modified. Also, the sanitization approach in Sedic may still reveal relative locations and length of sensitive data, which could lead to crucial information leakage in certain applications [25]. To overcome this weakness, [25] proposes tagged-MapReduce that augments each key-value pair with a sensitivity tag. Both solutions are not suitable for MapReduce job where all data is sensitive.

Encrypting part of dataset: In arguing that encrypting all data sets in cloud is not effective, [27] proposes an approach to identify which data sets with high frequency of accessing need to be encrypted while others are in cleartexts. This solution is not suitable for the case where all data have the same frequency of accessing or data owner does not want to reveal even a single tuple to untrusted cloud.

Other works **support very specific operations**. [7] searches encrypted keywords on the cloud without revealing any information about the content it hosts and search queries performed. [6] presents EPiC to count the number of occurrences of a pattern specified by user in an oblivious manner on the untrusted cloud. In contrast to these works, our work addresses more general problems, supporting any kind of operations.

2.2 Security in other Systems

Secure hardware at server side: Some works [5, 4] deploy the secure hardware at server side to ensure the confidentiality of the system. By leveraging server-hosted

tamper-proof hardware, [5] designs TrustedDB, a trusted hardware based relational database with full data confidentiality and no limitations on query expressiveness. However, TrustedDB does not deploy any parallel processing, limiting its performance. [4] also bases on the trusted hardware to securely decrypt data on the server and perform computations in plaintext. They present oblivious query processing algorithms so that an adversary observing the data access pattern learns nothing.

Secure hardware at client side. Even equipped with secure hardware on server, [5, 4] does not solve the two intrinsic problems of centralized approaches: (i) users get exposed to sudden changes in privacy policies; (ii) users are exposed to sophisticated attacks, whose cost-benefit is high on a centralized database [3]. So some works [21, 2, 3] are based on secure hardware at client side to solve these problems. The work in [2] proposes a generic Privacy-Preserving Data Publishing protocol, composed of low cost secure tokens and a powerful but untrusted supporting server, to publish different sanitized releases to recipients. Similarly, [21] proposes distributed querying protocols to compute general queries while maintaining strong privacy guarantees.

Centralized DaaS without secure hardware: Many works [18, 22] have addressed the security of outsourced database services (DaaS) by encrypting the data at rest and pushing part of the processing to the server side but none of them can achieve all aspects of security, utility, and performance. In terms of utility and security, the best theoretical solution such as fully homomorphic encryption [12], allows server to compute arbitrary functions over encrypted data without decrypting. However, this construction is prohibitively expensive in practice with overhead of $10^9\times$ [22]. In term of performance, CryptDB [18] provides provable confidentiality by executing SQL queries over encrypted data using a collection of efficient SQL-aware encryptions. But this system is not completely secure since it still uses some weak encryptions (e.g., deterministic & order-preserving encryptions [8]). Similarly, MONOMI system [22] securely executes arbitrarily complex queries over sensitive data on an untrusted database server with a median overhead of only 1.24× compared to an un-encrypted database. However, this system still uses some weak encryption schemes (e.g., deterministic encryption) to perform some SQL operations (e.g., Group By, equi-join).

As a conclusion, and to the best of our knowledge, no state-of-the-art MapReducre works can satisfy the three requirements of security, utility, performance, and our work is the first MapReduce-based proposal, that inherits the strong privacy guarantees from [21], achieving a secure solution to process large-scale encrypted data using a large set of tamper-resistant hardware with low performance overhead.

3 Context of the Study

3.1 Architecture

The architecture we consider is decentralized by nature. As pictured in Fig. 2, each individual is assumed to manage her data by means of a Trusted Data Server embedded in a secure device. We make no assumption about how this data is actually gathered and refer the reader to other papers addressing this issue [1, 17]. We detail next the main components of the architecture.

The Trusted Data Servers (TDSs). A TDS (as defined in [1]) is a DBMS engine embedded in an individual's secure device. It manages the individual's personal data and can participate in distributed queries while enforcing access control rules and opt-in/out choices of the individual. A TDS inherits its security from the Secure Device hosting it. Despite the diversity of existing hardware solutions, a Secure Device can be abstracted by (1) a Trusted Execution Environment and (2) a (potentially untrusted) mass storage area. E.g., the former can be provided by a tamper-resistant microcontroller while the latter can be provided by Flash memory (see Fig. 2). Since Secure Devices exhibit high security guarantee [1], the code executed by them cannot be tampered. This given, the contents of the mass storage area can be protected using cryptographic protocols. Most Secure Devices provide modest computing resources (see section 6) due to the hardware constraints linked to their tamper-resistance. On the other hand, a dedicated cryptographic co-processor usually handles cryptographic operations very efficiently (e.g., AES and SHA). Hence, even if there exist differences among Secure Devices, all provide *much stronger security guarantees* combined with a *much weaker computing power* than any traditional server.

The MapReduce Server. Due to their limited capacity, TDSs need a powerful Supporting Server running MapReduce framework to provide communication, intermediate storage and global processing services that TDSs cannot provide on their own. Being implemented on regular server(s), e.g., in the Cloud, mappers/reducers exhibit these properties: (1) Low Security, and (2) High Computing Resources.

3.2 Threat Model

TDSs are the unique element of trust in the architecture and are considered *honest*. Part of the Map and Reduce code embedded in TDSs is also assumed to be trusted. No trust assumption needs to be made on the querier either because (1) TDSs will not accept to participate to queries sent by a querier with insufficient privileges and (2) the querier can gain access only to the final result of the query computation (not to the raw data), as in traditional database systems. Preventing inferential attacks by combining the result of a sequence of authorized queries as in statistical databases and PPDP work is orthogonal to this study.

The potential adversary is consequently the mappers/reducers. We consider *honest-but-curious* mappers/reducers (i.e., which try to infer any information they can but strictly follows the protocol). Considering *malicious* mappers/reducers (i.e., which may tamper the protocol with no limit, including denial-of-service) is of little interest to this study. Indeed, a malicious mappers/reducers is likely to be detected with an irreversible political/financial damage and even the risk of a class action.

4 Proposed Solutions

4.1 MapReduce Job Execution Phases

The MapReduce programming model, depicted in Figure 3, consists of a map(k_1; v_1) function and a reduce(k_2; list(v_2)) function. The map(k_1; v_1) function is invoked for

every key-value pair $<k_1; v_1>$ in the input data to output zero or more key-value pairs of the form $<k_2; v_2>$. The reduce(k_2; list(v_2)) function is invoked for every unique key k_2 and corresponding values list(v_2) in the map output. reduce(k_2; list(v_2)) outputs zero or more key-value pairs of the form $<k_3; v_3>$. The MapReduce programming model also allows other functions such as (i) partition(k_2), for controlling how the map output key-value pairs are partitioned among the reduce tasks, and (ii) combine(k_2; list(v_2)), for performing partial aggregation.

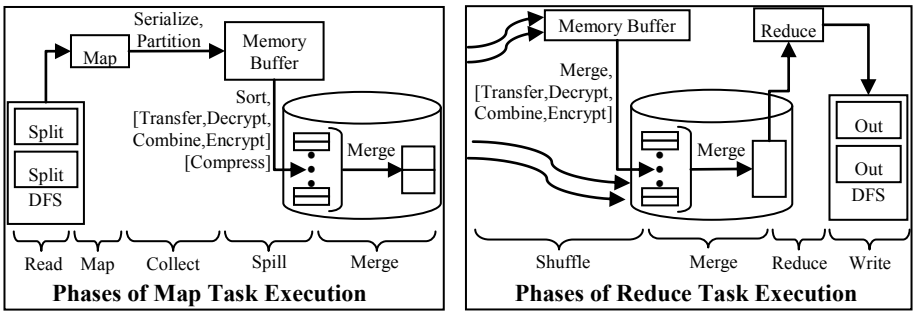


Fig. 3. Detail execution of map and reduce task

In the next section, we propose a solution so that we do not need to modify this original model. We use the encryption scheme to allow the untrusted mappers/reducers participate in the computation as much as possible and transfer the necessary computations that cannot be processed on server to TDSs. These transfer and computation on TDSs happen in parallel to speed up the running time.

4.2 Proposed Solution

Our proposed solution, called *ED_Hist*, builds on previous histogram-based techniques [20, 21] to prevent inferential attacks over encrypted data. Informally speaking, to prevent the frequency-based attack on deterministic encryption (*dEnc* for short) that encrypts the same cleartexts into the same ciphertexts, and to allow untrusted server group and sort the encrypted tuples (that have the same plaintext values) into the same partitions, *ED_Hist* transforms the original distribution of grouping attributes, called A_G, into a *nearly equi-depth histogram* (due to the data distribution, we cannot have exact equi-depth histogram). A nearly equi-depth histogram is a decomposition of the A_G domain into buckets holding *nearly* the same number of true tuples. Each bucket is identified by a hash value giving no information about the position of the bucket elements in the domain. Figure 4.a shows an example of an original distribution and Figure 4.b is its nearly equi-depth histogram.

There are three benefits in using nearly equi-depth histogram: i) allow mappers/reducers participate in the computation as much as possible (i.e., except the combine and reduce operations, all other operations can be processed in ciphertexts), without modifying the existing MapReduce framework; ii) better balance the load among mappers/reducers for skewed dataset; and iii) prevent frequency-based attack.

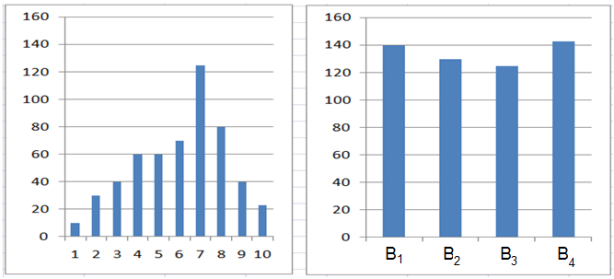


Fig. 4. Example of nearly equi-depth histogram

The protocol is divided into three tasks (see Figure 3, 5).

Collection Task: Each TDS allocates its tuple(s) to the corresponding bucket(s) and sends to mappers/reducers tuples of the form (F(k), nEnc(u)) where F is the mapping function that maps the keys to corresponding buckets:

$$bucketId = F(k)$$

and *nEnc* is the non-deterministic encryption that can encrypt the same cleartext into different ciphertext. Assume the cardinality of k is n, and F maps this domain to b buckets, then we have:

$$B_1 = F(k_{11}) = F(k_{12}) = \ldots = F(k_{1d})$$
$$B_2 = F(k_{21}) = F(k_{22}) = \ldots = F(k_{2e})$$
$$\ldots$$
$$B_b = F(k_{b1}) = F(k_{b2}) = \ldots = F(k_{bz})$$

From that, the average number of distinct plaintext in each bucket is: $h=(d+e+\ldots+z)/b=n/b$

When this task stops, all the encrypted data sent by TDSs are stored in DFS, and are ready to be processed by mappers/reducers.

Map Task: This task is divided into five phases:

1. Read: Read the input split from DFS and create the input key-value pairs: $(B_1, nEnc(u_1)), (B_2, nEnc(u_2)), \ldots (B_b, nEnc(u_m))$.

2. Map: Execute the user-defined map function to generate the map-output data: $map(B_i; nEnc(u_i)) \rightarrow (B'_i; nEnc(v_i))$. If the map function needs process complex functions that cannot be done on encrypted data (i.e., $v_i = f(u_i)$), connections to TDSs will be established to process these encrypted data.

3. Collect: Partition and collect the intermediate (map-output) data into a buffer before spilling.

4. Spill: Sort, if the combine function is specified: parallel transfer encrypted data to TDSs to decrypt, combine, encrypt, and return to mappers, perform compression if specified, and finally write to local disk to create file spills.

5. Merge: Merge the file spills into a single map output file. Merging might be performed in multiple rounds.

Reduce Task: This task includes four phases:

1. Shuffle: Transfer the intermediate data from the mapper nodes to a reducer's node and decompress if needed. Partial merging and combining may also occur during this phase.

2. Merge: Merge the sorted fragments from the different mappers to form the input to the reduce function.

3. Reduce: Execute the user-defined reduce function to produce the final output data. Since the reduce function can be arbitrary, and therefore encrypted data cannot be executed in reducers, they must be transferred to TDSs to be decrypted, executed the reduce function, encrypted, and returned to reducers. The difference between the output of the reduce function of traditional MapReduce with TrustedMR is that each input key represents different cleartext values, so the output key of the reduce function also represents different values: $(B'_1; list(nEnc(v_1)) \rightarrow (nEnc(k_{11}); nEnc(f(v_{11}))), ..., (nEnc(k_{1d}); nEnc(f(v_{1m})))$.

4. Write: Compressing, if specified, and writing the final output to DFS.

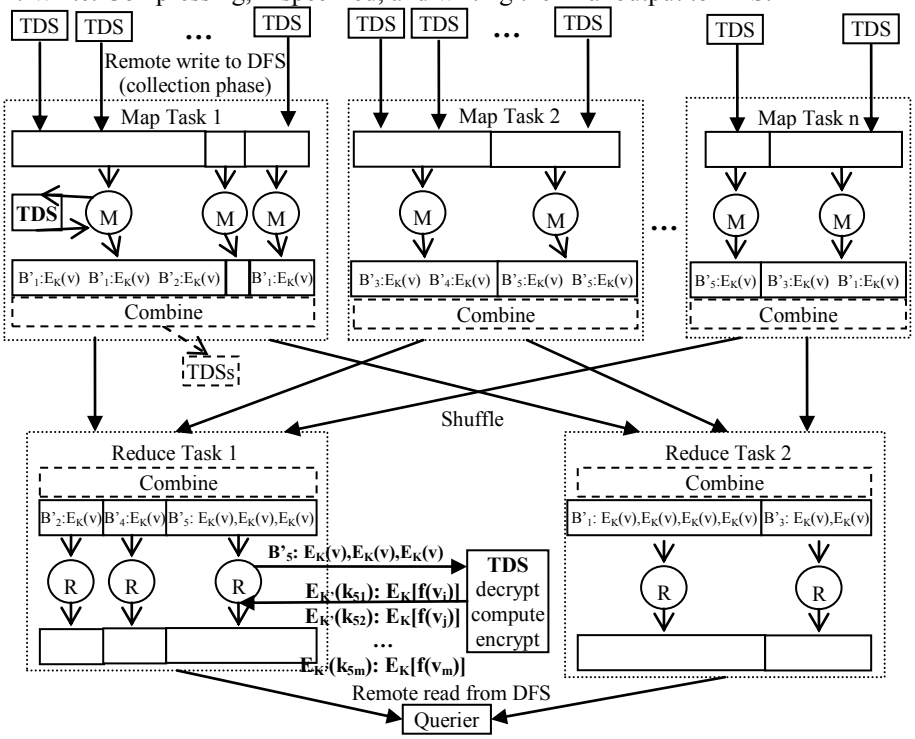


Fig. 5. Proposed solution

Among all phases in both map and reduce tasks, with the plaintext data mapped using the *ED_Hist*, the existing MapReduce framework can be used without being modified because each mappers/reducers can do all operations (i.e, map, partition, collect, sort, compress, merge, shuffle) on the mapped data, except the combine and reduce function. Since the combine and reduce functions must process on cleartexts, encrypted data are transferred back to TDSs for decrypting, computing, encrypting the result and returning to mappers/reducers. To reduce the overhead of transferring large amount of data between TDSs and mappers/reducers, each mappers/reducers split the

data into smaller pieces and send it in parallel to multiple TDSs. With this way, the transferring time is reduced. Fig. 6 is the pseudocode for map/reduce function.

```
method Map (bucket B_i; encrypted value nEnc(u_i))
1. emit(bucket B'_i, nEnc(v_i))

method Combine (bucket B'_i; list [nEnc(v_1), nEnc(v_2),..])
1. form the partition: nEnc(v_1), nEnc(v_2),..nEnc(v_p)
2. create connection and send data to TDSs
3. in each TDS:
4.     unmap bucket: F^-1(B'_i) -> k_i1 , k_i2, .., k_in
5.     decrypt nEnc(v_i) -> v_i
6.     compute r_ij = f(v_i) having the same k_ij
7.     encrypt result r_ij -> nEnc(r_ij)
8.     map to bucket: F(k_i1) = F(k_i2) =..= F(k_in) = B'_i
9. emit (bucket B'_i; nEnc(r_ij))

method Reduce (bucket B'_i; list [nEnc(r_ij),..])
1-7. similar to Combine function from step 1 to 7
8.   emit (nEnc(k_ij); nEnc(r'_ij))
```

Fig. 6. Map, Combine, and Reduce methods

Note that it is not possible to do the whole map and reduce tasks within TDS because the modest computing resource of TDS does not allow deploying the Hadoop. Also, data transfer between mappers/reducers and TDS are mandatory to keep the Hadoop framework unchanged. So, low power TDSs cannot do more than contributing to the internal execution of the map and reduce tasks.

4.3 How our proposed solution meets the requirements

Informally speaking, the security, utility and efficiency of the protocol are as follows (we formally prove the efficiency and security in the next sections):

Security. Since TDSs map the attributes to nearly equi-depth histogram, mappers/reducers cannot launch any frequency-based attack. What if mappers/reducers acquire a TDS with the objective to get the cryptographic material (i.e., a sort of collusion attack between mappers/reducers and a TDS)? As stated in section 3, TDS code cannot be tampered, even by its holder. Whatever the information decrypted internally, the only output that a TDS can deliver is a set of encrypted tuples, which does not represent any benefit for mappers/reducers.

Performance. The efficiency of the protocol is linked to the parallel computing of TDSs. Both the collection task and combine, reduce operations are run in parallel by all connected TDSs and no time-consuming task is performed by any of them. As the experiment section will clarify, each TDS manages incoming partitions in streaming because the internal time to decrypt the data and perform the computation is significantly less than the time needed to download the data. By combining the parallel computing, streaming data, and the crypto processor that can handles cryptographic operations efficiently in TDSs, our distributed model has acceptable and controllable performance overhead as pointed out in experiment.

Generality. Since the data is processed by trusted TDSs in cleartext, our solution can support any form of Map and Reduce functions.

Seamless integration: Because we do not need to modify the original MapReduce framework, our solution can easily integrate with the existing framework. ED_Hist helps mappers/reducers run on encrypted data exactly as if they run on cleartext data without modifying the original MapReduce framework (i.e., as pointed out in section 4.2, the only tasks that mappers/reducers cannot run on encrypted data are combine and reduce).

Beside the four essential requirements above, we can easily show that our solution provides also a correct and exact result. Since mappers/reducers are honest-but-curious, it will strictly follow the protocol and deliver to the querier the final output. Unlike the differential privacy, mappers/reducers do not sanitize the output (to achieve the differential privacy), so the final output is exact. If a TDS goes offline in the middle of processing a partition, and therefore cannot return result as expected, mappers/reducers will resend that partition to another available TDS after waiting the response from disconnected TDS a specific interval.

5 Privacy Analysis

5.1 Security of Basic Encryption Schemes

In cryptography, indistinguishability under chosen plaintext attack (IND-CPA) [30] (which is proved to be equivalent to semantic security [14]) is a very strong notion of security for encryption schemes, and is considered as a basic requirement for most provably secure cryptosystems. While *nDet_Enc* is believed to be IND-CPA [32], *Det_Enc*, on the other hand, cannot achieve semantic security or indistinguishability due to lack of randomness in ciphertext. The maximum level of security for *Det_Enc* that can be guaranteed is PRIV [31] which is a weaker notion of security than IND-CPA. Then, it is important to understand how much (quantitatively) less secure the *Det_Enc* and *ED_Hist* are, in compare with *nDet_Enc*. To address this question, we use the *coefficient* to measure the security level of *Det_Enc* and *ED_Hist*, given the *nDet_Enc* as the highest bound of security level.

5.2 Information Exposure with Coefficient

In this section, in order to quantify the confidentiality of each encryption scheme, we measure the information exposure of the encrypted data they reveal to SSI by using the approach proposed in [10] which introduces the concept of coefficient to assess the exposure. To illustrate, let us consider the example in Fig. 7 where Fig. 7a is taken from [10] and Fig. 7b is the extension of [10] applied in our context. The plaintext table Accounts is encrypted in different ways corresponding to encryption schemes. To measure the exposure, we consider the probability that an attacker can reconstruct the plaintext table (or part of the table) by using the encrypted table and his prior knowledge about global distributions of plaintext attributes.

Although the attacker does not know which encrypted column corresponds to which plaintext attribute, he can determine the actual correspondence by comparing their cardinalities. Namely, she can determine that I_A, I_C, and I_B correspond to

attributes Account, Customer, and Balance respectively. Then, the IC table (the table of the inverse of the cardinalities of the equivalence classes) is formed by calculating the probability that an encrypted value can be correctly matched to a plaintext value. For example, with *Det_Enc*, $P(\alpha = Alice) = 1$ and $P(\kappa = 200) = 1$ since the attacker knows that the plaintexts *Alice* and *200* have the most frequent occurrences in the Accounts table (or in the global distribution) and observes that the ciphertexts α and κ have highest frequencies in the encrypted table respectively. The attacker can infer with certainty that not only α and κ represent values *Alice* and *200* (*encryption inference*) but also that the plaintext table contains a tuple associating values *Alice* and *200* (*association inference*). The probability of disclosing a specific association (e.g., *<Alice,200>*) is the product of the inverses of the cardinalities (e.g., $P(<\alpha,\kappa> = <Alice,200>) = P(\alpha = Alice) \times P(\kappa = 200) = 1$). The *exposure coefficient* $\mathcal{E}$ of the whole table is estimated as the average exposure of each tuple in it:

$$\varepsilon = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{k} IC_{i,j}$$

Here, n is the number of tuples, k is the number of attributes, and $IC_{i,j}$ is the value in row i and column j in the IC table. Let's N_j be the number of distinct plaintext values in the global distribution of attribute in column j (i.e., $N_j \leq n$).

ACCOUNTS

Account	Customer	Balance
Acc1	Alice	500
Acc2	Alice	200
Acc3	Bob	300
Acc4	Chris	200
Acc5	Donna	400
Acc6	Elvis	200

DETERMINISTIC ENCRYPTION

Enc_tuple	I_A	I_C	I_B
x4Z3tfX2ShOSM	π	α	μ
mNHg1oC010p8w	ω	α	κ
WsIaCvfyF1Dxw	ξ	β	η
JpO8eLTVgwV1E	ψ	γ	κ
qctG6XnFNDTQc	ϕ	δ	θ
4QbqC3hxZHklU	Γ	ε	κ

IC TABLE OF DETERMINISTIC ENCRYPTION

ic_A	ic_C	ic_B
1/6	1	1/3
1/6	1	1
1/6	1/4	1/3
1/6	1/4	1
1/6	1/4	1/3
1/6	1/4	1

a

NON-DETERMINISTIC ENCRYPTION

I_A	I_C	I_B
π	λ	μ
ω	α	χ
ξ	β	η
ψ	γ	κ
ϕ	δ	θ
Γ	ε	τ

EQUI-DEPTH HISTOGRAM

I_A	I_C	I_B
π	α	μ
π	α	κ
π	β	μ
ξ	β	κ
ξ	δ	μ
ξ	δ	κ

IC TABLE OF NON-DETERMINISTIC ENCRYPTION

ic_A	ic_C	ic_B
1/6	1/5	1/4
1/6	1/5	1/4
1/6	1/5	1/4
1/6	1/5	1/4
1/6	1/5	1/4
1/6	1/5	1/4

b

Fig. 7. Encryption and IC tables

As pointed out above, the encrypted centralized databases [22] use *Dec_Enc* that opens the door for frequency-based attack. However, when using *nDet_Enc*, the more secure encryption scheme than *Det_Enc*, it cannot help MapReduce framework process encrypted data since mappers/reducers cannot group and sort the same encrypted tuples into the same partition. Equi-depth histogram overcomes the weakness of these two schemes.

Using *nDet_Enc*, because the distribution of ciphertexts is obfuscated uniformly, the probability of guessing the true plaintext of α is P(α = *Alice*) = 1/5. So, $IC_{i,j} = 1/N_j$ for all i, j, and thus the exposure coefficient of *nDet_Enc* is:

$$\varepsilon_{nDec\_Enc} = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{k}\frac{1}{N_j} = 1/\prod_{j=1}^{k}N_j$$

For the nearly equi-depth histogram, each hash value can correspond to multiple plaintext values. Therefore, each hash value in the equivalence class of multiplicity *m* can represent any *m* values extracted from the plaintext set, that is, there are $\binom{N_j}{m}$ different possibilities. The identification of the correspondence between hash and plaintext values requires finding all possible partitions of the plaintext values such that the sum of their occurrences is the cardinality of the hash value, equating to solving the NP-Hard *multiple subset sum problem* [9]. We consider two critical values of collision factor *h* (defined as the ratio G/M between the number of groups G and the number M of distinct hash values) that correspond to two extreme cases (i.e., the least and most exposure) of ε_{ED_Hist}: (1) $h = G$: all plaintext values collide on the same hash value and (2) $h = 1$: distinct plaintext values are mapped to distinct hash values (i.e., in this case, the nearly equi-depth histogram becomes *Det_Enc* since the same plaintext values will be mapped to the same hash value).

In the first case, the optimal coefficient exposure of histogram is:

$$\min(\varepsilon_{ED\_Hist}) = 1/\prod_{j=1}^{k}N_j$$

because $IC_{i,j} = 1/N_j$ for all i, j. For the second case, the experiment in [9] (where they generated a number of random databases whose number of occurrences of each plaintext value followed a Zipf distribution) varies the value of *h* to see its impact to ε_{ED_Hist}. This experiment shows that the smaller the value of *h*, the bigger the ε_{ED_Hist} and ε_{ED_Hist} reaches maximum value (i.e., $\max(\varepsilon_{ED\_Hist}) \approx 0.4$) when $h = 1$.

The exposure coefficient gets the highest value when no encryption is used at all and therefore all plaintexts are displayed to attacker. In this case, $IC_{i,j} = 1\ \forall$ i, j, and thus the exposure coefficient of plaintext table is (trivially):

$$\varepsilon_{P\_Text} = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{k}1 = 1$$

In short, *ED_Hist* is more secure than *Det_Enc*, and at some point the *ED_Hist* can get the same high security as *nDet_Enc*. Specifically, if all plaintext values collide on the same mapped value, *ED_Hist* has the least exposure, similar to *nDet_Enc*. On the contrary, if distinct plaintext values are mapped to distinct hash values, *ED_Hist* exposes the most amount information to server (i.e., in this case, the nearly equi-depth histogram becomes *Det_Enc* since the same plaintext values will be mapped to the same value).

The information exposures among our proposed solutions are summarized in Fig. 8. In conclusion, the information exposures of *nDet_Enc, Det_Enc* and *ED_Hist* have the following order: $\varepsilon_{nDec\_Enc} \leq \varepsilon_{ED\_Hist} \leq \varepsilon_{Dec\_Enc} < 1$, meaning that *ED_Hist* is the intermediate between *nDet_Enc* and *Det_Enc*.

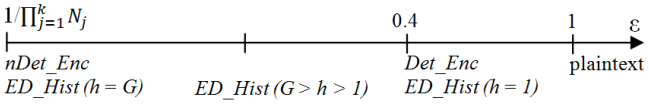


Fig. 8. Information exposure among encryption schemes

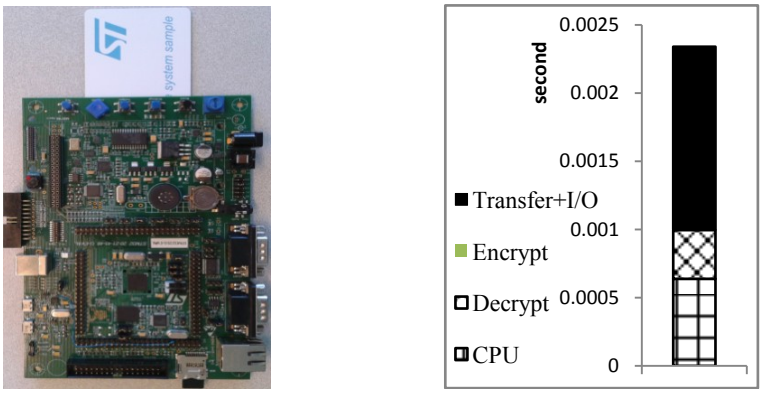
6 Performance Evaluation

This section evaluates the performance of our solution. By nature, the behavior of secure devices is difficult to observe from the outside and integrating performance probes in the embedded code significantly changes the performance. To circumvent this difficulty, we first perform tests on a development board running the same embedded code (including the operating system RTOS) and having the same hardware characteristics (same microcontroller and Flash storage) as our secure devices. This gave us the detail time breakdown on the secure hardware (i.e., transfer, I/O, crypto, and CPU cost). Then we use the Z-token described below to test on the larger scale (i.e., running multiple Z-tokens in parallel) in the real cluster. We also compare the running time on ciphertext and that on cleartext to see how much overhead incurred. We finally increase the power of the cluster by scaling depth (i.e., increase the number of Z-tokens plugged in each node) and scaling width (i.e., increase the number of nodes) to see the difference between the two ways of scaling.

6.1 Unit Test on Development Board

To see the detail time contributing to the total execution time on the secure hardware, we performed unit tests on the development board presented in Fig. 9a. This board has the following characteristics: the microcontroller is equipped with a 32 bit RISC CPU clocked at 120 MHz, a crypto-coprocessor implementing AES and SHA in hardware (encrypting or decrypting a block of 128bits costs 167 cycles), 64 KB of static RAM, 1 MB of NOR-Flash and is connected to a 1 GB external NAND-Flash and to a smartcard chip hosting the cryptographic material. The device can communicate with the external world through USB connection.

We measured on this device the performance of the main operations influencing the global cost, that is: encryption, decryption, communication and CPU time. Fig. 9b depicts this internal time consumption of this platform. The transfer cost dominates the other costs due to the connection latencies. The CPU cost is higher than cryptographic cost because (1) the cryptographic operations are done in hardware by the crypto-coprocessor and (2) TDS spends CPU time to convert the array of raw bytes (resulting from the decryption) to the number format for calculation later and some extra operations. Encryption time is much smaller than decryption time because only the result of the aggregation of each partition needs to be encrypted. TDSs handle data from mappers/reducers in stream due to the fact that encryption and CPU time is less than transfer time and I/O operations. So, TDSs can process the old data while receiving the new one at the same time.



a) b)

Fig. 9. Unit test on real hardware

6.2 Experimental Setup

Our experiment conducted with secure devices has been performed on a cluster of Paris Nord University. Each node is equipped with 4-core 3.1 GHz Intel Xeon E31220 processor, 8GB of RAM, and 128GB of hard disk. These nodes run on Debian Wheezy 7 with unmodified Hadoop 1.0.3. It is the Cloud provider who decides number of map/reduce tasks. The number of TDSs is also fixed by Cloud provider who plugs these tokens. The experiments will give hints how to choose the number of tokens and nodes. We run the Hadoop in parallel on ZED secure tokens (Fig. 10).

6.3 Scaling with Parallel Computing

Figure 12 shows the performance overhead when processing ciphertext over cleartext. There is no difference in map time but the reduce time in ciphertext is much longer than that of cleartext. This is due to the time to connect to Z-token and process the encrypted data inside the Z-token. In this test, only one Z-token is plugged to each node. That creates the bottleneck for the ciphertext processing because Z-token is much less powerful than the node that has to wait Z-token to process the encrypted data. While the cleartext data is processed directly in the powerful node, the ciphertext has to be transferred to tokens for processing. In this way, computation on ciphertext incurs three overhead in compared with the cleartext: i) time to transfer the data from node to token (including the connection time and I/O cost), ii) time to decrypt the data and encrypt the result, iii) the constraint on the CPU and memory size of token for computation inside the token.

To alleviate this overhead, we plug multiple tokens to the same node and process the ciphertext in parallel in these tokens. Figure 11 shows the 20 tokens run in parallel and plugged to the same node. In Figure 13, when the number of tokens plugged to each node increases, the reduce time decreases gradually and approaches that of cleartext. Specifically, when the number of tokens increases from 1 to 20, the average

speedup is 1.75. When we plug 32 tokens to each node, the reduce time reaches 5.49 (seconds), which gives approximate 10% longer than cleartext. Hence, the overhead is controllable by increasing the number of tokens plugged per reducer.

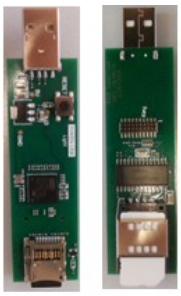


Fig. 10. ZED token (front and back sides)

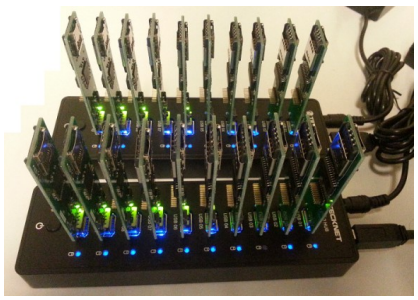


Fig. 11. Twenty tokens running in parallel

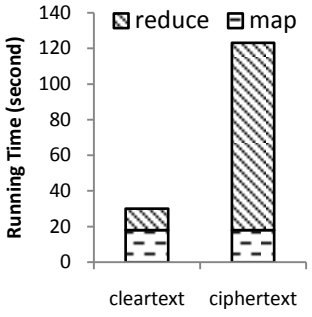


Fig. 12. Running time of cleartext & ciphertext

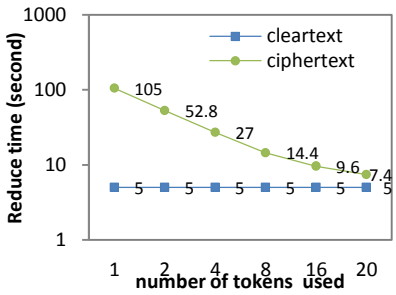


Fig. 13. Scaling depth

6.4 Scaling Depth versus Scaling Width

In traditional MapReduce, the cluster can be scaled depth by increasing number of processors per node or scaled width by increasing number of nodes. In TrustedMR, since the cluster depends on the tokens for cryptographic operations, we scale depth by increasing the number of tokens (i.e., from 1 to 4) plugged to each node. We also scale width by increasing number of nodes (i.e., from 1 to 4), and then we compare the two ways of scaling. In this test, we also increase the size of the dataset (i.e., from 2 million tuples to 4 million tuples) to see how the running time varies.

In Figure 14 & 15, when we increase the number of nodes in the cluster and keep the same number of tokens on each node, the reduce time decreases accordingly and vice versa. Also, with the same number of tokens, plugging them to the same node or to multiple nodes gives almost no difference in term of running time (e.g., the reduce time of 4 nodes with each node having only 1 token is only few percent difference from that of 1 node having 4 tokens plugged). Furthermore, the average speedup of scaling width is 1.74 which is only 2% different from that of scaling depth (i.e., 1.71). In conclusion, scaling depth yields nearly the same performance as scaling width. The

only factor that affects the overall performance of the cluster is the total number of tokens plugged to this cluster, no matter how they are distributed to each node. Based on this conclusion, we measure the performance with the configuration of 5 nodes having 20 tokens plugged in each node (i.e., 100 tokens in total) and use this measurement (together with the speedups measured above) to simulate the performance with larger scale and bigger dataset (at the moment, it is difficult to perform large scale experiments with smart tokens due to the hardware cost[4]). Figure 16 shows the performance with the 1TB dataset. When the number of nodes in the cluster increases (with the number of tokens plugged in each node fixed at 20), the running time reduces correspondingly. The time to process 1 TB data is acceptable (e.g., a few minutes) when we have enough nodes in the cluster.

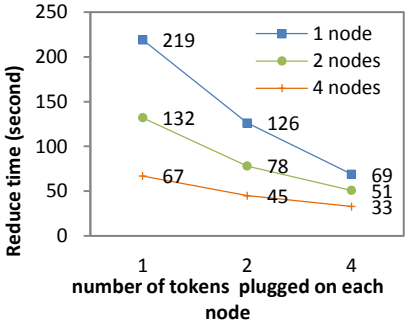


Fig. 14. Reduce time for 2 million tuples

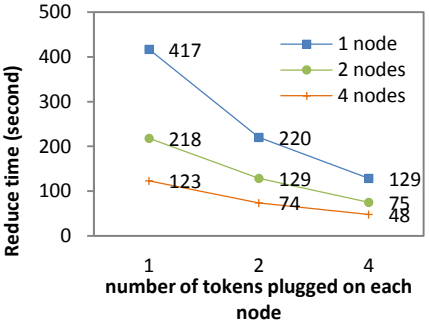


Fig. 15. Reduce time for 4 million tuples

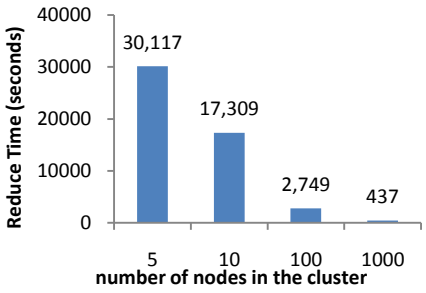


Fig. 16. Reduce time for 1TB dataset

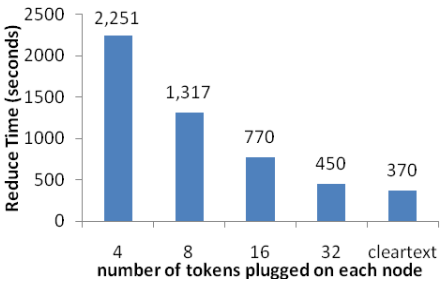


Fig. 17. Comparison of TrustedMR and [13]

To compare with state-of-the-art work in the literature, Figure 17 compares the reduce running time of TrustedMR with the Hadoop system running in cleartext in [13]. The experiment in [13] runs on a Hadoop cluster of 16 Amazon EC2 nodes of the c1.medium type with the 10GB dataset. We simulate our system with 16 nodes and vary the number of tokens to compare. It is easy to see that when the number of tokens plugged on each node increases, the overhead performance decrease thanks to the parallel computation of tokens. Although it is only a rough comparison, it gives

[4] We have only 100 tokens, so this is the possible maximum number of tokens we can use for the experiments.

the illustration that the performance overhead of TrustedMR is not very far (i.e., 1.2× longer) from the original Map Reduce program running in cleartext.

7 Conclusion

This paper proposed a new approach to process big personal data using MapReduce while maintaining privacy guarantees. It draws its novelty from the fact that (private) user data remains under the control of its owner, itself embedded in a secure enclave within the untrusted Cloud platform. Our solution meets four main requirements, namely security, performance, generality, and seamless integration. Our future work will (1) extend threat model to consider strong adversaries capable of compromising tamper-resistant devices and (2) perform comparisons with other server-based architectures exploiting secure hardware (e.g., IBM 4765 PCIe Crypto Coprocessor).

Acknowledgement

The authors wish to thank Nicolas Greneche from University of Paris Nord for his help in setting up the cluster for the experiment. This work is partially funded by project ANR-11-INSE-0005 "Keeping your Information Safe and Secure".

References

1. Allard, T., Anciaux, N., Bouganim, L., Guo, Y., Le Folgoc, L., Nguyen, B., Pucheral, P., Ray, Ij., Ray, Ik., Yin, S.: Secure Personal Data Servers: a Vision Paper. VLDB, pp. 25-35. Singapore (2010)
2. Allard, T., Nguyen, B., Pucheral, P.: MET_AP: Revisiting Privacy-Preserving Data Publishing using Secure Devices. DAPD, 2013.
3. Anciaux, N., Bonnet, P., Bouganim, L., Nguyen, B., Sandu Popa, I., Pucheral, P.: Trusted Cells: A Sea Change for Personal Data Services. CIDR, USA , 2013.
4. Arasu, A., Kaushik, R.: Oblivious Query Processing. ICDT 2014
5. Bajaj, S., Sion, R.: TrustedDB: a trusted hardware based database with privacy and data confidentiality. SIGMOD Conference 2011: 205-216
6. Blass, E., Noubir, G., Huu, T.V.: EPiC: Efficient Privacy-Preserving Counting for MapReduce. In IACR Cryptology ePrint Archive (2012) 452.
7. Blass, E. O., Pietro, R. D., Molva, R., Önen, M.: PRISM-Privacy-Preserving Search in MapReduce. In PETS, pp 180-200, 2012.
8. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-Preserving Symmetric Encryption. EUROCRYPT, pp 224-241, (2009).
9. Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Modeling and assessing inference exposure in encrypted databases. ACM TISSEC, vol 8(1), pp. 119-152, (2005)
10. Damiani, E., Capitani Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. CCS, pp. 93-102, (2003)
11. Dean, J., and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Commun. ACM, 51(1):107–113, 2008.

12. Gentry, C.: Fully homomorphic encryption using ideal lattices. STOC, pp. 169-178. (2009)
13. Herodotou, H., Babu, S.: Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. PVLDB 4(11): 1111-1122 (2011)
14. Goldwasser, S., and Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences, 28(2):270–299. 1984.
15. Lam, H.Y., Fung, G.S.K., and Lee, W.K.: A Novel Method to Construct Taxonomy Electrical Appliances Based on Load Signatures. IEEE Transactions on Consumer Electronics. 53(2), 653-660.2007.
16. Mun M., Hao S., Mishra N. et al., "Personal data vaults: a locus of control for personal data streams," in Proc. of the 6th Int. Conf on Emerging Networking Experiments and Technologies (Co-NEXT '10), New York, USA, December 2010.
17. de Montjoye, Y-A., Wang, S. S., Pentland, A.: On the Trusted Use of Large-Scale Personal Data. IEEE Data Eng. Bull. 35(4): 5-8 (2012)
18. Popa, R. A., Redfield, C. M. S., Zeldovich, N., et al. CryptDB: protecting confidentiality with encrypted query processing. In SOSP, pp 85–100, 2011.
19. Roy, I., Setty, S., Kilzer, A., Shmatikov, V., and Witchel, E.: Airavat: Security and privacy for MapReduce. USENIX NSDI, pp. 297–312, 2010.
20. Hacigumus, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over encrypted data in database service provider model. ACM SIGMOD, pp. 216-227. Wisconsin (2002)
21. To, Q.C., Nguyen, B., Pucheral, P.: Privacy-Preserving Query Execution using a Decentralized Architecture and Tamper Resistant Hardware, EDBT, pp. 487-498, 2014.
22. Tu, S., Kaashoek, M. F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. In PVLDB, pp 289-300, 2013.
23. Wei, W., Du, J., Yu, T., and Gu, X.: SecureMR: A Service Integrity Assurance Framework for MapReduce. ACSAC, pp. 73–82, 2009.
24. Zhang, K., Zhou, X., Chen, Y., Wang, X., Ruan, Y.: Sedic: privacy-aware data intensive computing on hybrid clouds. CCS 2011: 515-526
25. Zhang, C., Chang, E., Yap, R.: Tagged-MapReduce: A General Framework for Secure Computing with Mixed-Sensitivity Data on Hybrid Clouds. CCGrid, pp 31-40, 2014.
26. Zhang, X., Yang, L.T., Liu, C., Chen, J.: A Scalable Two-Phase Top-Down Specialization Approach for Data Anonymization Using MapReduce on Cloud. Parallel and Distributed Systems, IEEE Transactions on , vol.25, no.2, pp.363-373, 2014.
27. Zhang, X., Liu, C., Nepal, S., Pandey, S., and Chen, J.: A Privacy Leakage Upper-bound Constraint based Approach for Cost-effective Privacy Preserving of Intermediate Datasets in Cloud, IEEE Transactions on Parallel and Distributed Systems, 24(6): 1192-1202, 2013
28. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data. Official Journal of the EC, 23, 1995.
29. Wang, J., Wang, Z. A Survey on Personal Data Cloud. *The Scientific World Journal*, 2014.
30. Katz, J., and Lindell, Y.: Introduction to Modern Cryptography: Principles and Protocols. Chapman and Hall/CRC. 2007.
31. Bellare, M., Boldyreva, Alexandra., and O'Neill, Adam.: Deterministic and efficiently searchable encryption. In CRYPTO. Lecture Notes in Computer Science, volume 4622. 535–552. 2007.
32. Arasu, A., Eguro, K., Kaushik, R., and Ramamurthy, R.: Querying Encrypted Data (Tutorial). In ACM SIGMOD Conference. 2014.